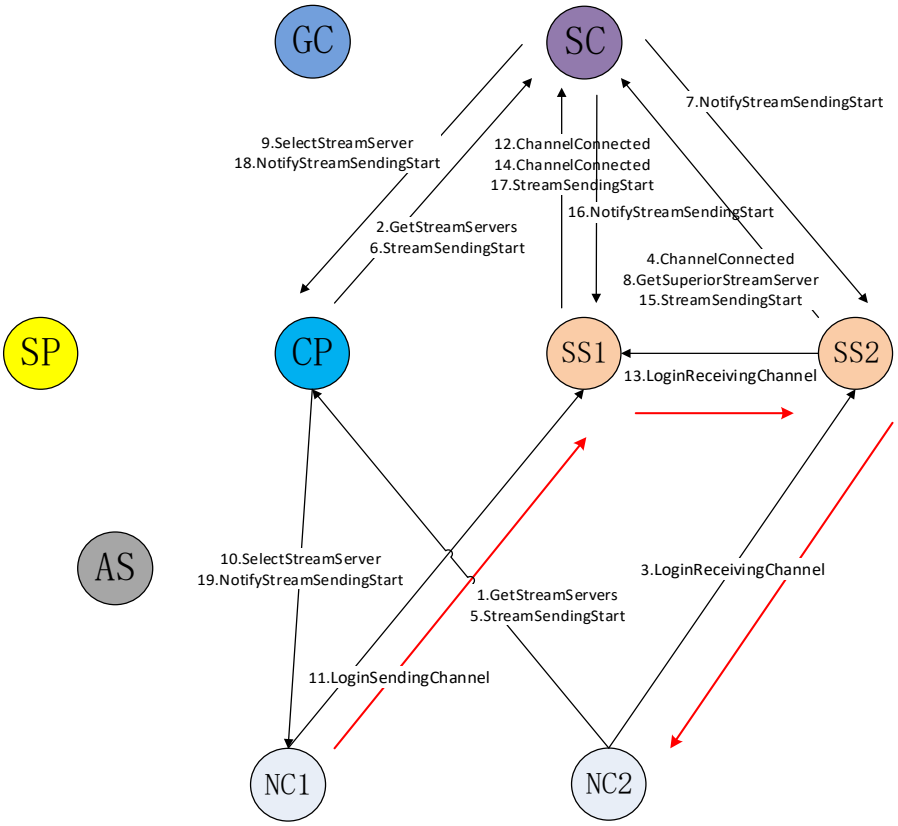
zookeeper 实现 SC 分布式锁代码说明

一、 需求背景

本代码主要是解决流式服务中多个媒体服务器同时请求获得上一级节点时被多个 SC 收到并处理而引起数据一致性问题。具体说明如下：



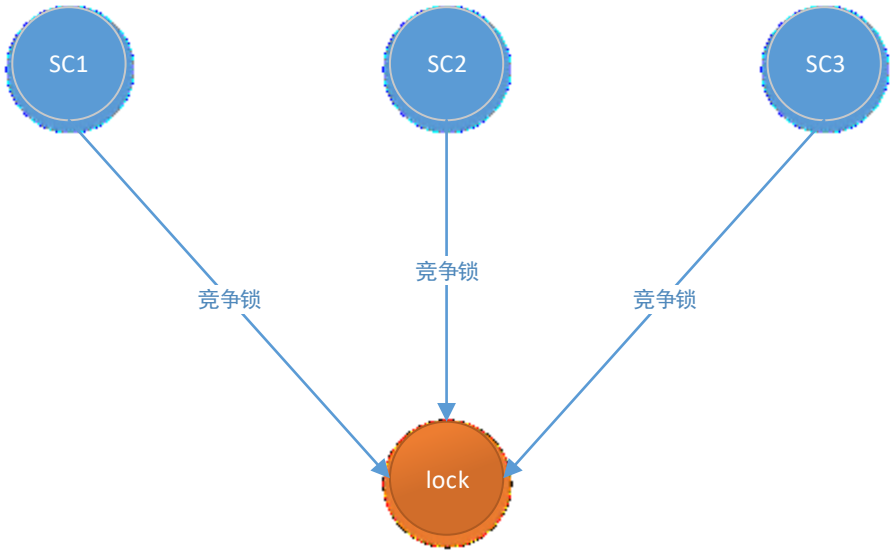
如上图第 8 步，SC 收到 GetSuperiorStreamServer 请求，会调用 ice 查询请求方的上一级节点，如果数据库中没有，则调用 ice 查询流的源节点，如果成功查到，则路径选择一台 SS 作为请求方的上一级，返回 GetSuperiorStreamServer 请求，如果流的源节点没有选择，则像流的发布端发起选择流源节点的请求，既上图的第 9 步，然后在上图的第 12 步做路径选择，然后返回 GetSuperiorStreamServer 请求。整个过程不仅要多次读写数据库，还有可能经过中间很多步骤才能完成这个接口的完整调用。如果不做数据一致性处理，如果多个 SS 同时请求该接口，发布端可能会多次选择发布源节点。

二、 设计思路

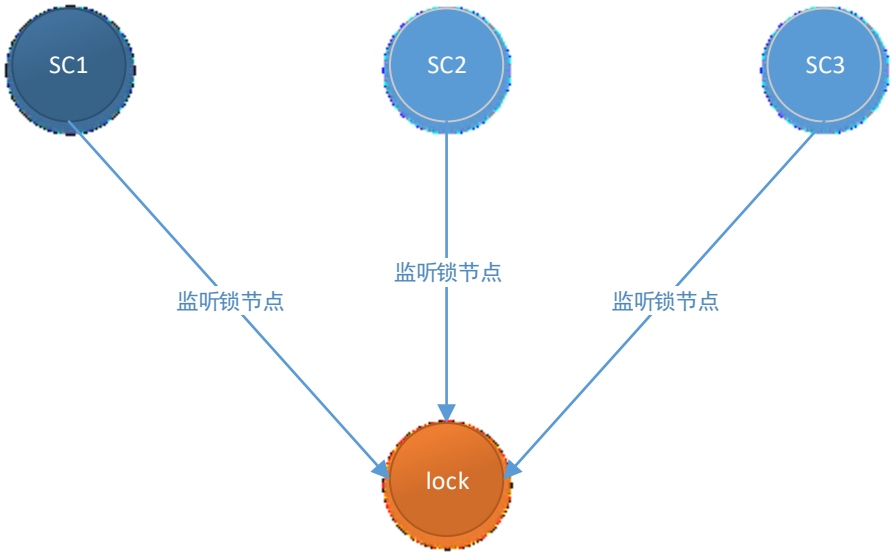
严格意义上说，这里的分布式锁不是通用的分布式锁，这里的场景只需要解决同一时刻多个 SS 调用 SC 中 GetSuperiorStreamServer 接口时有且只有一个 SC 处理该请求，其他 SC 的请求保存在本地，等到获得锁的 SC 处理完释放锁后统一回复保存的请求即可。具体设计如下：

2.1 竞争锁

1、多个 SC 并发调用获得上一级节点时：

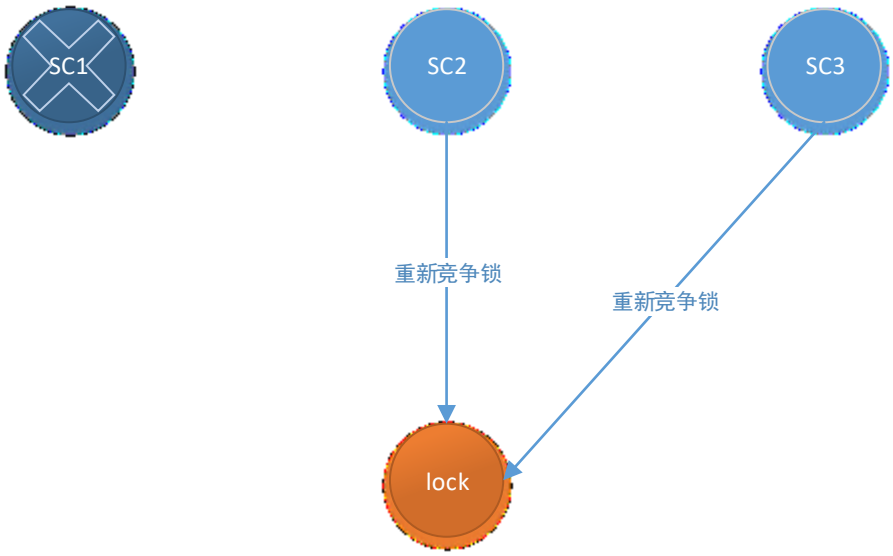


2、SC1 获得锁，SC2，SC3 删除自己创建的节点，并监听锁节点的删除事件。

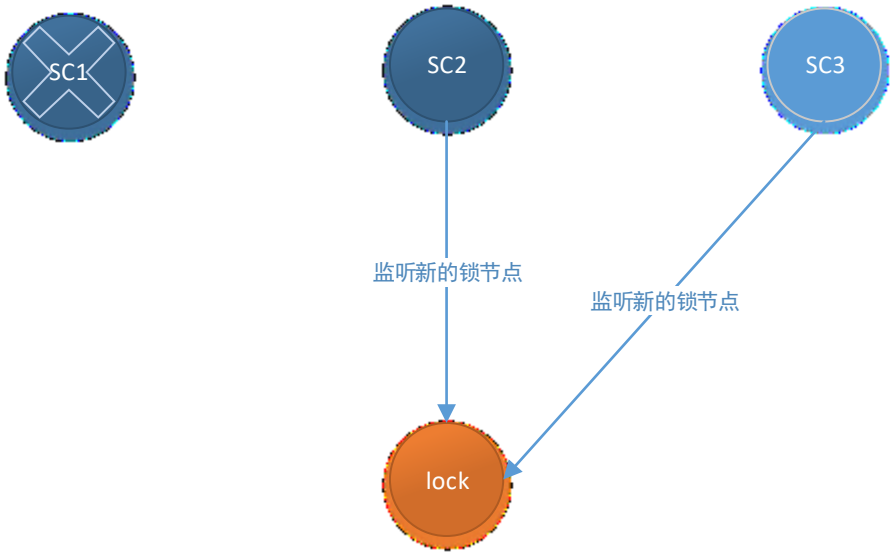


2.2 获得锁节点挂掉

1、锁节点挂掉后，SC2，SC3 监听到锁节点删除事件，重新竞争锁

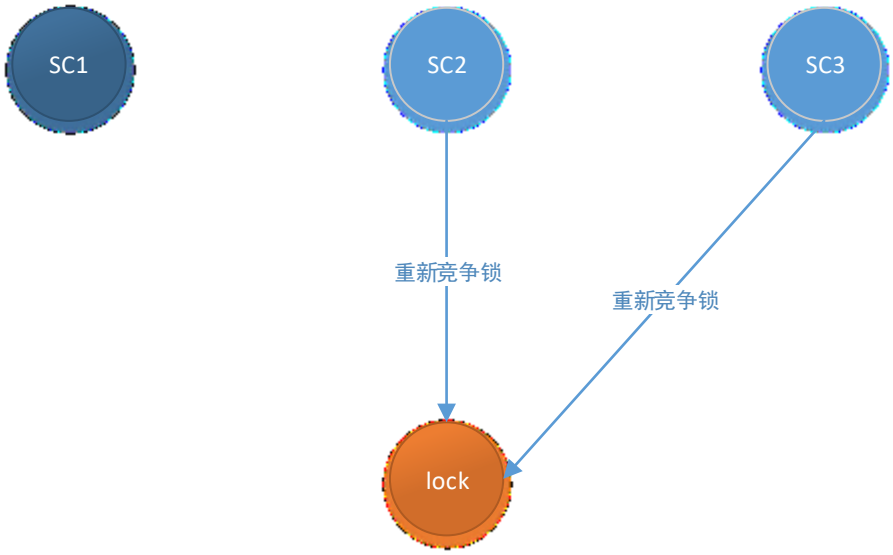


2、SC2 获得锁后，SC2，SC3 删除之前的监听对象，重新监听新的锁节点的删除事件

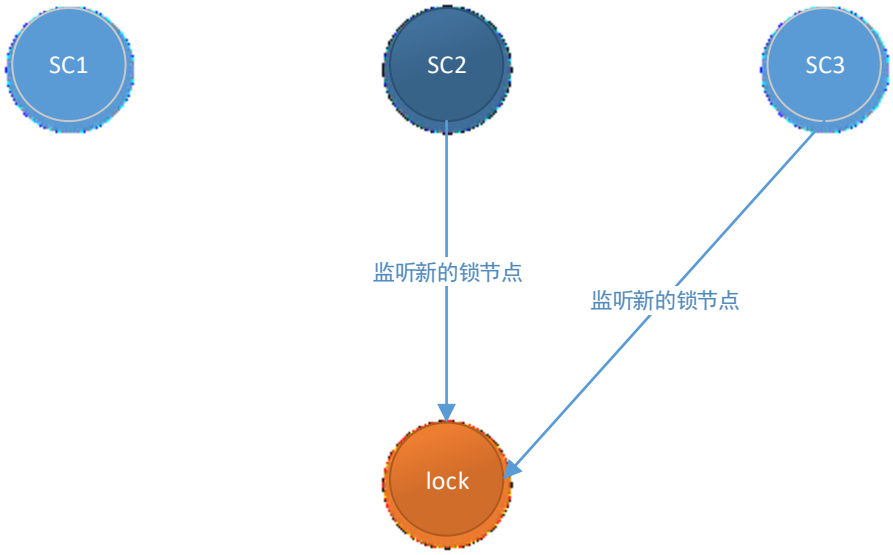


2.3 获得锁节点与 zookeeper 断开连接

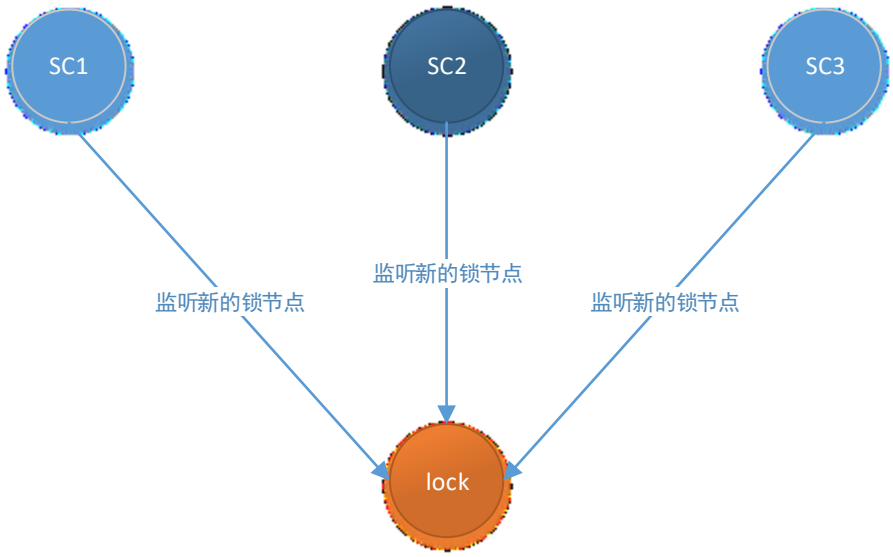
1、获得锁的 SC1 与 zookeeper 断开连接时，锁节点被删除，SC1 删除之前的锁节点的监听对象，SC2，SC3 监听到锁节点的删除事件，然后 SC2，SC3 重新竞争锁



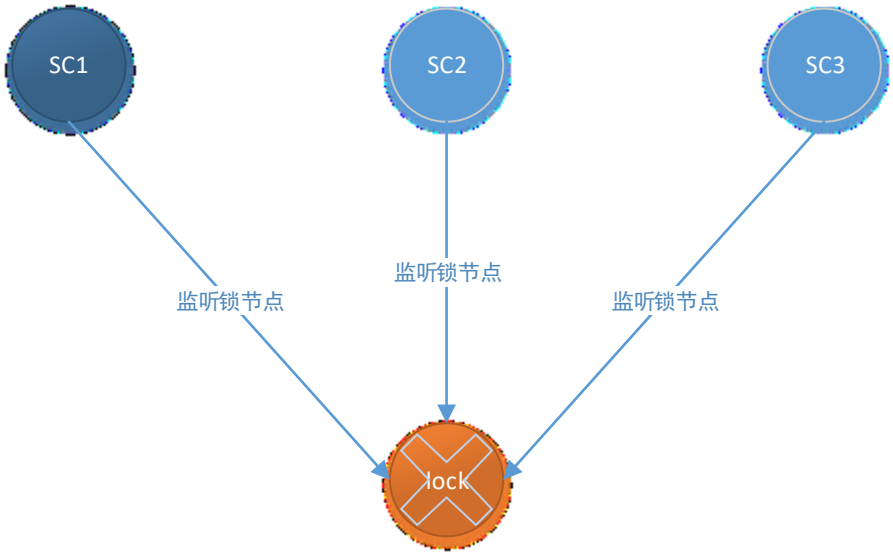
2、SC2 竞争到锁，SC2，SC3 删除之前的监听对象，重新监听新的锁节点的删除事件



3、SC1 与 zookeeper 重新建立连接时，SC1 监听新的锁节点



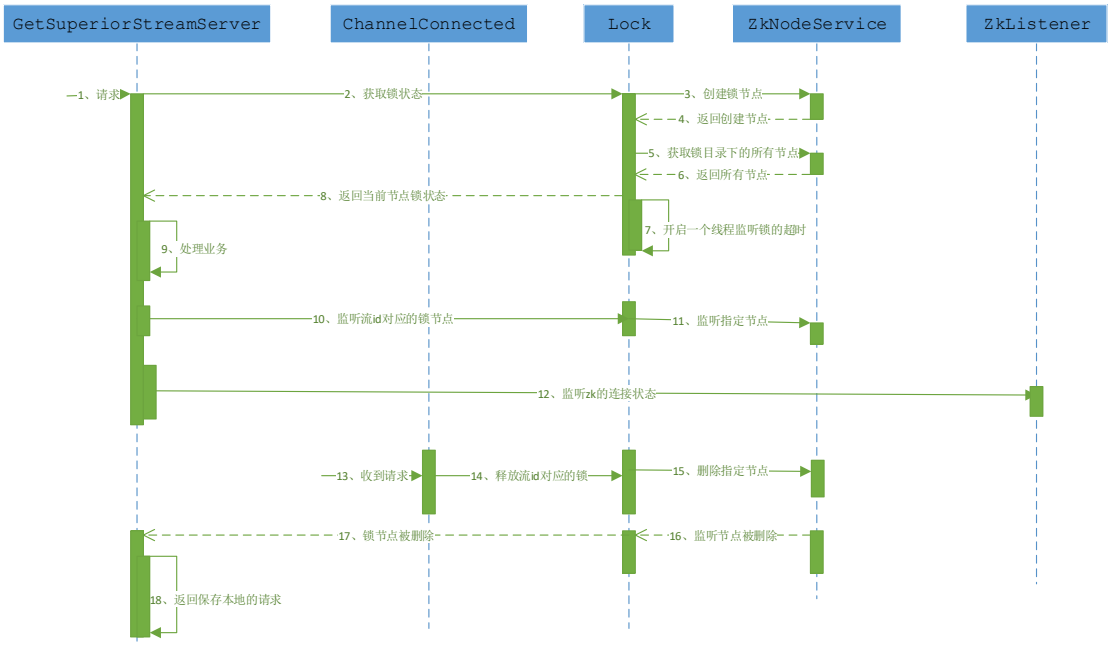
2.4 锁节点被删除



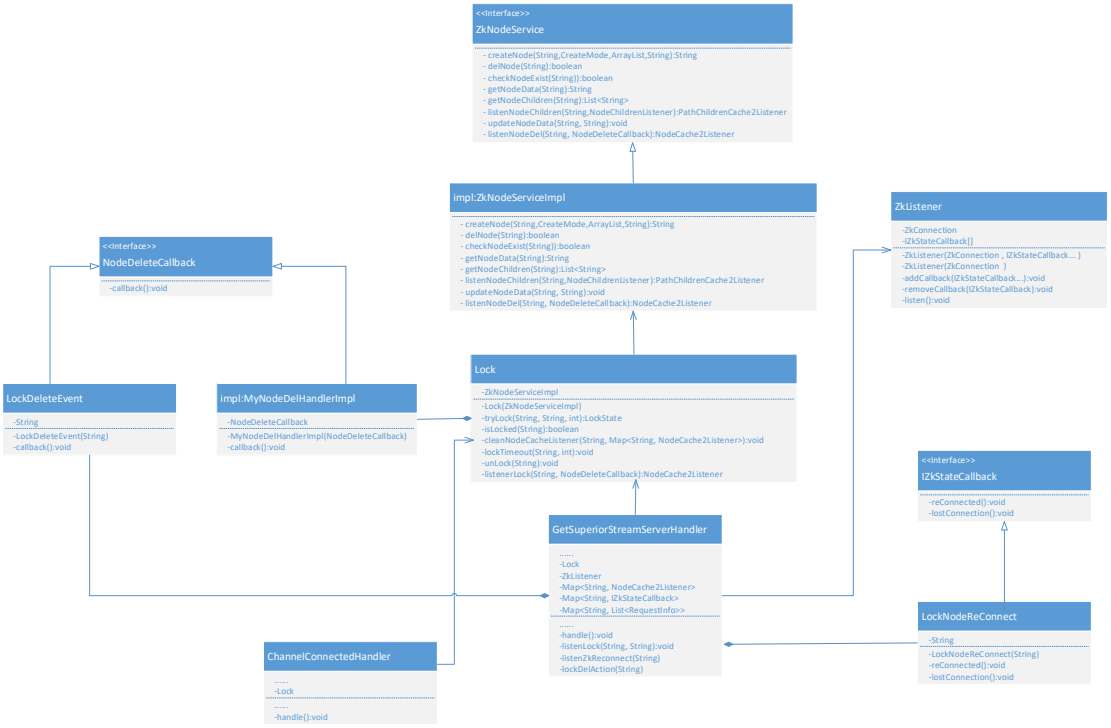
说明：锁节点被删除只有三种情况，一，该接口正常调用完，二，获得锁节点挂掉，三，获得锁的节点与 zookeeper 断开连接。但是该接口的关键点是选择源节点，只要写操作只能是获得锁的节点执行，其他节点只能读即可。如果锁节点被删除，所有监听锁的节点都会收到删除事件，然后判断源节点是否已经选择，如果源节点已经选择，说明是正常执行完，回复获得上一级节点请求。如果源节点没有选择，则说明出现异常情况，则重新竞争锁，重新选择源节点即可。

三、 实现说明

1. 业务处理时序图：



2. 实现设计



四、代码文件

1、 ZkNodeService.java：操作 zookeeper 节点的接口

2、 ZkNodeServiceImpl.java：操作 zookeeper 节点的接口实现类

3、 ZkListener.java：监听 zookeeper 连接状态的处理类

4、 Lock.java：锁的实现类
5、 LockState.java：本节点的锁的状态
6、 NodeDeleteCallback.java：锁节点被删除的回调接口
7、 IZkStateCallback.java：zookeeper 状态改变的回调接口
8、 GetSuperiorStreamServerHandler.java：需要加锁的接口
9、 ChannelConnectedHandler.java：释放锁的接口

五、总结

优点：

1. 面向接口编程，功能模块解耦。
2. 功能分层，类职责单一。
3. 使用回调函数，将业务功能与公共功能分开，便于业务功能灵活拓展。
4. 公共模块不使用 spring 等框架，业务模块使用 spring 框架，将公共模块的功能自动注入进来，消除公共模块第三方框架的依赖。
5. 媒体服务器和锁节点都使用超时处理，避免死锁和保证每个请求都能回复。

缺点：

每次调用该接口都必须加锁后，才能处理消息，有性能消耗。